Build, Test, Ship, Learn, Rinse & repeat.

07 October 2015 · agile, kanban, learning, scrum, system

Ever feel like your team is in a deadlock? The product owner wants Gizmo+ to be shipped, your senior engineers are split between grokking Gizmo+ and fixing Widget++. Meanwhile the SDETs (http://blogs.msdn.com/b/seliot/archive/2010/04/18/what-is-an-sdet.aspx) are frantically updating automated checks/BDD scripts and exploratory testers are uncovering that Widget+ and Gizmo+ should have been named ...+10 given the number of surprise bonus features they are finding. As a consequence feature delivery can start to slow and quality is inevitably hit as difficult decisions are made on what to fix.

The typical reactions to such a situation can depend on your project's context, but to highlight a few common ones:

- Ramp up team size.
- Push back on deadlines.
- Push back on new features.
- Delay releases until 'it all gets sorted' ...

I don't have to break it to you that these options are 'far from optimal'. In summary they all revolve around costing more and delivering less (from my time as a programme manager I can tell you - thats what we call a hard sell).

I won't claim there is a 'magic bullet', because - there isn't. But lets try and break the problem down a little:

1. We can't seem to get enough stuff out the door
2. Our definition of 'enough stuff' seems to be growing
3. What stuff does get 'done', often needs 'redoing'
4. GO TO (1)

Speak to your team, and you'll probably find a common feeling among the team is that they are 'overwhelmed' and 'have to much to do'. Hence the instinct many people have to just 'ramp up the team'. But you can interpret those statements in two ways. Another way to see the problem is, they 'have too much to do [all at once]'.

A good analogy is the dishwasher. For example, We don't have a large kitchen, so we have a small or to be more precise: a slimline dish washer. Its a top of the range model complete with a confusing array of space age controls, and an ability to run extra quiet. But it doesn't have enough space to fit a whole days worth of dishes, cups etc. Now I could 'ramp up' to a bigger dishwasher. But thats going to get messy and expensive I'll have start plumbing in a new

washer, remodeling the kitchen - replacing other equipment and fittings that work just fine right now.

So My next option is to casually disregard the sensible arrangement of dishes suggested in the manual, and load the dishes in to something approaching the density of super-dense collapsing star. As you might imagine, the cleaning ability of the dishwasher is somewhat reduced. And a lot of the dishes end up needing to be re-done in the morning. Whats even worse, I can't easily tell which dishes were cleaned - and which are providing a nutritious environment for bacteria. I have to awkwardly examine each dish in turn. As each dish was crammed in at the last minute in an undocumented free-for-all - I can't even reliably automate some tests/checks to help with this.

I lose many of the benefits of my machine as well, Its no-longer quiet and efficient, hugging a tree, healing my karma and cleaning my dishes in one go. Instead, I have to switch it to the options that translate roughly as 'noisy and inefficient' and 'i hate the planet'.

The following day, I of course have a slightly larger pile of dishes to clean. The new days dishes and the failure-demand (https://en.wikipedia.org/wiki/Failure_demand) I inherited from the day before. Just like your real feature releases, this is more work and reputational harm for your team.

Why not run the dishwasher twice? After breakfast, Fill the system to a level that seems to deliver and run it. If you haven't quite filled all the slots, thats not a problem - run it anyway. Why? because you are evening out the flow of dishes. By under-loading the dishwasher in the morning, you don't have to over-load it in the evening. Inspecting the product becomes quicker and easier to do.

This approach provides a steady cadence to your engineers and testers. The tsunami of each big cycle becomes a more manageable ebb and flow. Getting those fixes and features out cleanly and regularly helps provide regular feedback to the team.

Did we develop something wrong? or miss a bug? we'll know sooner and can fix our team to stop it happening again [sooner]. Leave it a while, and soon the list of missed and broken things becomes something to have long and painful meetings about.

Frequent releases, can help deliver this calmer, more stable development and test process, where some features are delivered sooner and the team isn't trying to build a large complicated system, with every release. They can focus on developing and testing a small set of changes, against the back drop of a relatively stable system.